assi

A) Base path + quick pre-check

```
echo"$USER"# shows your Linux username (used in paths)
BASE="/home/$USER/directus-junior-assessment"# set the base folder path
echo"$BASE"# confirm the full base path
mkdir -p"$BASE"# create the base folder if missing
cd"$BASE"# go to the working folder
pwd# confirm you are inside /home/<user>/directus-junior-assessment
```

B) Install all required tools (Ubuntu/Debian)

B1) System packages

```
sudo apt-get update# refresh package lists
sudo apt-get install -y \
  ca-certificates curl gnupg lsb-release \
  zip unzip git jq openssl# install basics + jq + openssl
```

B2) Install Docker Engine + Docker Compose plugin

```
sudo install -m 0755 -d /etc/apt/keyrings# create keyring folder
curl -fsSL https://download.docker.com/linux/ubuntu/gpg \
  |sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg# add Docker GPG key
sudochmod a+r /etc/apt/keyrings/docker.gpg# allow apt to read key

echo"deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/do
cker.gpg] \
https://download.docker.com/linux/ubuntu$(. /etc/os-release && echo $VERSI
ON_CODENAME) stable" \
|sudotee /etc/apt/sources.list.d/docker.list > /dev/null# add Docker repo

sudo apt-get update# refresh package lists again (Docker repo added)
sudo apt-get install -y \
  docker-ce docker-ce-cli containerd.io \
  docker-buildx-plugin docker-compose-plugin# install docker + compose
```

B3) Run docker without sudo

```
sudo usermod -aG docker"$USER"# add your user to docker group
newgrp docker# reload group membership in current terminal
docker version# verify docker works
docker compose version# verify compose works
```

If `newgrp docker` doesn't work: log out + log in, then run `docker version`.

C) Create project folders (FULL PATH)

```
mkdir -p"$BASE"/{submission,screenshots,data/postgres,uploads,extension
s}# create all folders
ls -la"$BASE"# confirm folders exist
```

D) Create `.env` (admin login + keys) (FULL PATH)

D1) Create admin email/password

```
cat >"$BASE/.env" <<'ENV'
ADMIN_EMAIL=admin@example.com
ADMIN_PASSWORD=ChangeThisToAStrongPassword123!
DIRECTUS_URL=http://localhost:8055
ENV
```

D2) Generate Directus KEY + SECRET and append to `.env`

```
KEY=$(openssl rand -hex 32)# generate random KEY
SECRET=$(openssl rand -hex 32)# generate random SECRET
echo"KEY=$KEY" >>"$BASE/.env"# write KEY into .env
echo"SECRET=$SECRET" >>"$BASE/.env"# write SECRET into .env
cat"$BASE/.env"# confirm .env contents
```

E) Create `docker-compose.yml` (FULL PATH)

```
cat >"$BASE/docker-compose.yml" <<'YAML'
services:
  db:
    image: postgres:16-alpine
    container_name: directus_db
    restart: unless-stopped
    environment:
      POSTGRES_USER: directus
      POSTGRES_PASSWORD: directuspass
      POSTGRES_DB: directus
    volumes:
      - /home/${USER}/directus-junior-assessment/data/postgres:/var/lib/postg
resql/data
```

```
    ports:
      -"5432:5432"
    healthcheck:
test: ["CMD-SHELL","pg_isready -U directus -d directus"]
      interval: 5s
timeout: 3s
      retries: 20

  directus:
    image: directus/directus:11
    container_name: directus_app
    restart: unless-stopped
    depends_on:
      db:
        condition: service_healthy
    env_file:
      - /home/${USER}/directus-junior-assessment/.env
    environment:
      KEY:"${KEY}"
      SECRET:"${SECRET}"
      ADMIN_EMAIL:"${ADMIN_EMAIL}"
      ADMIN_PASSWORD:"${ADMIN_PASSWORD}"

      DB_CLIENT:"pg"
      DB_HOST:"db"
      DB_PORT:"5432"
      DB_DATABASE:"directus"
      DB_USER:"directus"
      DB_PASSWORD:"directuspass"

# optional but helpful
      PUBLIC_URL:"http://localhost:8055"

    ports:
      -"8055:8055"
    volumes:
      - /home/${USER}/directus-junior-assessment/uploads:/directus/uploads
      - /home/${USER}/directus-junior-assessment/extensions:/directus/extensi
ons
YAML
```

✅ Note: I used **absolute paths** inside the YML exactly as you requested.

F) Start Directus (local only)

```
cd"$BASE"# go to the project folder
docker compose pull# download images (directus + postgres)
docker compose up -d# start in background
docker compose ps# confirm both containers are running
docker logs -n 80 directus_app# check Directus boot logs (last 80 lines)
```

F1) Verify Directus responds

```
curl -s http://localhost:8055/server/ping ;echo# quick “pong” check (if availab
le)
curl -s http://localhost:8055/ |head -n 5# fetch homepage HTML start
```

Open in browser:

- `http://localhost:8055`

Login with:

- `ADMIN_EMAIL` and `ADMIN_PASSWORD` from `"$BASE/.env"`

✅ Take screenshot **with localhost visible** and admin logged in.

G) Create Data Model (Collections + Fields + Validation)

Go Directus UI: **Settings → Data Model**

G1) Collection: `events` (fields EXACTLY per assessment)

Create collection: **events**

Add fields:

1. `title`

- Type: **String**
- Validation: **Required**

1. `description`

- Type: **Text**
- Optional

1. `event_date`

- Type: **DateTime**
- Validation: **Required**

1. `location`

- Type: **String**
- Optional

1. `capacity`

- Type: **Integer**
- Validation: **Required**, **Min = 1**

1. `status`

- Type: **Dropdown**
- Values: `draft`, `published`, `cancelled`

1. `created_at`

- Type: **Timestamp**
- Default: **Current timestamp** / auto create

✅ Screenshot: `collections.png` after all collections are created.

G2) Collection: `categories`

Create collection: **categories**

Fields:

1. `name`

- Type: **String**
- Validation: **Required** + **Unique**

1. `slug`

- Type: **String**
- Requirement: **Auto-generated from name**

How to auto-generate slug:

- Edit field `slug` → choose **Interface: Slug** (or “Text Input” with slug generation if your UI shows it)
- Set **Generate from = name**
- Optional: set **Lowercase = ON**, **Replace spaces = '-'**

G3) Collection: `registrations`

Create collection: **registrations**

Fields:

1. `name`

- Type: **String**
- Validation: **Required**

1. `email`

- Type: **String**
- Validation: **Required** + **Email format**

1. `event`

- Type: **Relationship (Many-to-One)**
- Link to: **events**
- Required

1. `registered_at`

- Type: **Timestamp**
- Default: **Current timestamp** / auto create

G4) Junction table: `events_categories` (Many-to-Many)

Create relationship:

- In `events` collection → **Add Field → Relationship → Many-to-Many**
- Related collection: `categories`
- Junction collection: **events_categories** (create it)

This gives:

- `events ↔ categories` many-to-many via `events_categories`

✅ Screenshot after you see the relationship field in events.

G5) Add required calculated field for Flow #2

In `events`, add new field:

- `spots_remaining`
  - Type: **Integer**
  - Default: optional 0 (you can leave blank; flow will update)

H) Permissions & Security (3 roles)

Go: **Settings → Roles & Permissions**

H1) Admin role

- Must authenticate via login
- Give **Full CRUD** on `events`, `categories`, `registrations`, `events_categories`

✅ Screenshot: `permissions-admin.png`

H2) Create role: `Event Manager`

Permissions EXACTLY per assessment:

- `events`: **Create, Read, Update** (NO Delete)
- `categories`: **Read only**
- `registrations`: **Read, Delete** (NO Create/Update)

✅ Screenshot: `permissions-manager.png`

H3) Public role (IMPORTANT filter)

Permissions EXACTLY per assessment:

- `events`: **Read only** but **only where status = published**
- `categories`: **Read only**
- `registrations`: **Create only**

Public events READ filter (permission filter JSON):

```
{"status":{"_eq":"published"}}
```

✅ Screenshot: `permissions-public.png`

I) Flow #1 — Capacity Enforcement (BLOCK registrations when full)

Go: **Settings → Flows → Create Flow**

Name: `Block Registration When Full`

Trigger

- Trigger type: **Event Hook**
- Collection: **registrations**
- Event: **Before Create**
- Mode: **Blocking** (must block)

Step 1: Read event capacity

Operation: **Read Data**

- Collection: `events`
- Read: **Single Item**
- Key: use variable picker → `{{$trigger.payload.event}}`
- Output name: `event_item`

Step 2: Count existing registrations for that event

Operation: **Read Data**

- Collection: `registrations`
- Read: **Aggregate**
- Aggregate: **Count** of `id`
- Filter: event equals trigger event

  Filter JSON (use variable picker for the event id):

```
{"event":{"_eq":"{{ $trigger.payload.event }}"}}
```

- Output name: `reg_count`

Step 3: Condition (IF count >= capacity → throw error)

Operation: **Condition**

- If: `reg_count.count >= event_item.capacity`

Step 4: Throw error (inside TRUE branch)

Operation: **Throw Error**

- Message: `Event is fully booked`

✅ Screenshot: `flow-capacity.png`

I1) Demo test (UI quick)

1. Create event `capacity = 2`
2. Create 2 registrations
3. Try 3rd registration → must fail with `Event is fully booked`

✅ Take screenshot showing both **success** and **rejected** state.

J) Flow #2 — Auto-calc `spots_remaining`

Because delete payload can be tricky, this is the **safe “no error” approach**:

We’ll make **3 flows**.

J1) Flow 2A: After Create registration → recompute spots

Name: `Recalc Spots Remaining (After Create Registration)`

Trigger:

- Event Hook
- Collection: `registrations`

- Event: **After Create**

Steps:

1. Read event (`events` item id = `{{$trigger.payload.event}}`)
2. Aggregate count registrations for that event
3. Calculate `spots_remaining = capacity - count`
4. Update event item (set `spots_remaining`)

J2) Flow 2B: Before Delete registration → recompute "after delete" result

Name: `Recalc Spots Remaining (Before Delete Registration)`

Trigger:

- Event Hook
- Collection: `registrations`
- Event: **Before Delete**
- Mode: **Blocking** (so we can read the record before it disappears)

Steps:

1. Read the registration being deleted
    - Read Data → `registrations` single item
    - Key: `{{$trigger.keys[0]}}`
    - Output: `reg_item`
2. Read event (`events` id = `{{ reg_item.event }}`)
3. Count registrations for that event (this count includes the one being deleted)
4. Compute remaining **after delete**:
    - `spots_remaining = capacity - (count - 1)`
5. Update the event with computed spots_remaining

J3) Flow 2C: After Update event capacity → recompute spots

Name: `Recalc Spots Remaining (After Update Event)`

Trigger:

- Event Hook
- Collection: `events`
- Event: **After Update**

Steps:

1. Read updated event: id = `{{$trigger.keys[0]}}`
2. Count registrations for that event
3. spots_remaining = capacity - count
4. Update event spots_remaining

✅ Screenshot: `flow-calculate.png`

K) Import categories (CSV) + create sample data

K1) Create CSV file (FULL PATH)

```
cat >"$BASE/categories.csv" <<'CSV'
name,slug
Technology,technology
Workshop,workshop
Networking,networking
Conference,conference
Social,social
CSV
```

In Directus UI:

- Go to `categories` collection
- Click **Import**
- Upload: `"$BASE/categories.csv"`
- Confirm records imported

✅ Screenshot categories list.

K2) Create at least 3 events (with M2M categories)

Create 3 events:

- Assign categories (M2M)
- Make **at least one** event `status = published`
- Set different dates and capacities

✅ Screenshot events list and one event edit page showing categories.

L) UI Configuration (required)

In `events` collection:

- Configure list columns: `title`, `event_date`, `status`, `spots_remaining`
- Default sort: `event_date`
- Improve form layout (group fields, reorder, set widths)
- Add icons if you want

✅ Screenshot: `ui-config.png`

M) Export `directus-export.json` (COMMAND METHOD, no UI needed)

M1) Login via API and get token

```
source"$BASE/.env"# load ADMIN_EMAIL / ADMIN_PASSWORD / DIRECTUS_
URL / KEY / SECRET into shell

TOKEN=$(curl -s -X POST"$DIRECTUS_URL/auth/login" \
  -H"Content-Type: application/json" \
  -d"{\"email\":\"$ADMIN_EMAIL\",\"password\":\"$ADMIN_PASSWORD\"}" \
  | jq -r'.data.access_token')# login and extract access_token

echo"$TOKEN" |head -c 20 ;echo"..."# confirm token looks valid (don’t share f
ull token)
```

M2) Download schema snapshot to required file path

```
mkdir -p"$BASE/submission"# ensure submission folder exists

curl -s -H"Authorization: Bearer $TOKEN" \
"$DIRECTUS_URL/schema/snapshot" \
  >"$BASE/submission/directus-export.json"# export snapshot JSON

ls -lh"$BASE/submission/directus-export.json"# confirm file exists and size >
0
```

N) Bonus: Duplicate email prevention per event (easy + strong)

N1) Add UNIQUE index on registrations(event, email)

Directus UI:

- Data Model → `registrations` → **Indexes**

- Create **Unique** index with fields: `event` + `email`

This gives the bonus (+5) and prevents duplicates at DB level.

(Optionally add a Flow to throw a nicer message, but UNIQUE index is enough.)

O) Create ERD diagram (command way)

O1) Install Node + Mermaid CLI (Ubuntu/Debian)

```
sudo apt-get install -y nodejs npm# install node + npm (simple method)
node -v# confirm node installed
npm -v# confirm npm installed

sudo npm i -g @mermaid-js/mermaid-cli# install mermaid renderer CLI
mmdc -h |head -n 3# confirm mermaid cli works
```

O2) Create ERD Mermaid file (FULL PATH)

```
cat >"$BASE/submission/erd.mmd" <<'MMD'
erDiagram
  events ||--o{ registrations : has
  events }o--o{ categories : categorized_as

  events {
    intid
    string title
    text description
    datetime event_date
    string location
    int capacity
    string status
    timestamp created_at
    int spots_remaining
  }

  categories {
    intid
    string name
    string slug
  }

  registrations {
    intid
    string name
    string email
    int event
    timestamp registered_at
  }
MMD
```

O3) Render ERD image (FULL PATH)

```
mmdc -i"$BASE/submission/erd.mmd" -o"$BASE/submission/erd.png"# gene
rate ERD PNG
ls -lh"$BASE/submission/erd.png"# confirm ERD file created
```

P) Create README.pdf (required)

P1) Create README.html (FULL PATH)

```
cat >"$BASE/submission/README.html" <<'HTML'
<!doctype html>
<html>
<head>
```

```
  <meta charset="utf-8">
  <title>Directus Assessment - README</title>
  <style>
    body { font-family: Arial, sans-serif; margin: 40px; line-height: 1.4; }
    h1,h2 { margin-top: 22px; }
    code, pre { background:#f4f4f4; padding: 2px 6px; border-radius: 4px; }
    img { max-width: 100%; border: 1px solid#ddd; padding: 8px; }
  </style>
</head>
<body>
<h1>Directus Junior Assessment Submission</h1>

<h2>Setup Instructions</h2>
<ul>
  <li>Directus 11.x runs locally via Docker Compose.</li>
  <li>Startcommand: <code>docker compose up -d</code></li>
  <li>URL: <code>http://localhost:8055</code></li>
</ul>

<h2>ERD Diagram</h2>
<p>Collections: events, categories, registrations, junction events_categories.
</p>
<img src="erd.png" alt="ERD">

<h2>Data Model Explanation</h2>
<ul>
  <li><b>events</b> stores event details including capacity and status.</li>
  <li><b>registrations</b> links to events (M2O) and stores attendee data.</li
>
  <li><b>categories</b> supports tagging events (M2M).</li>
</ul>

<h2>Permissions Explanation</h2>
<ul>
  <li><b>Admin</b>: full CRUDfor all collections.</li>
  <li><b>Event Manager</b>: create/read/update events,read categories,rea
```

```
d/delete registrations.</li>
  <li><b>Public</b>:read published events only,read categories, create regist
rations only.</li>
</ul>

<h2>Flows Explanation</h2>
<ul>
  <li><b>Capacity enforcement</b>: blocking before create registration; coun
ts registrations; rejectsif full.</li>
  <li><b>Auto-calculate</b>: updates events.spots_remaining on create/delet
e registrations and capacity update.</li>
</ul>

<h2>Challenges Faced</h2>
<p>(Fill this section with any issues you hit and how you solved them.)</p>

</body>
</html>
HTML
```

P2) Convert HTML → PDF (easy)

```
sudo apt-get install -y wkhtmltopdf# install HTML-to-PDF tool
cd"$BASE/submission"# go to submission folder (so erd.png resolves)
wkhtmltopdf README.html README.pdf# generate README.pdf
ls -lh"$BASE/submission/README.pdf"# confirm PDF exists
```

Q) Put screenshots in exact required structure

```
mkdir -p"$BASE/submission/screenshots"# create screenshots folder
```

Save screenshots with these names (exact):

- `local-setup.png`
- `collections.png`
- `permissions-admin.png`
- `permissions-manager.png`
- `permissions-public.png`
- `flow-capacity.png`
- `flow-calculate.png`
- `ui-config.png`

R) Final submission folder check + ZIP

R1) Verify final structure

```
cd"$BASE"# go to base folder
find"$BASE/submission" -maxdepth 2 -type f |sort# list submission files
```

Expected (minimum):

- `$BASE/submission/directus-export.json`
- `$BASE/submission/README.pdf`
- `$BASE/submission/erd.png`
- `$BASE/submission/screenshots/*`

R2) Create ZIP (FULL PATH)

```
cd"$BASE"# ensure we zip from base
zip -r"$BASE/submission.zip" submission# create zip file
ls -lh"$BASE/submission.zip"# confirm zip created
```

S) Troubleshooting (common problems + exact commands)

S1) Port 8055 already used

```
sudo lsof -i :8055# see what is using port 8055
docker compose down# stop Directus if running
# If something else uses it, stop that service OR change ports in docker-comp
ose.yml
```

S2) Directus container keeps restarting

```
docker logs -n 200 directus_app# view errors (most common: DB connection)
docker logs -n 200 directus_db# view postgres errors
docker compose ps# see container health
```

S3) Reset everything (DANGER: deletes DB)

```
cd"$BASE"# go to project folder
docker compose down -v# stop and remove volumes (ERASE DB)
rm -rf"$BASE/data/postgres"# delete postgres data folder
docker compose up -d# recreate fresh DB + directus
```

S4) Backup DB quickly

```
mkdir -p"$BASE/backups"# create backup folder
dockerexec -t directus_db pg_dump -U directus directus >"$BASE/backups/di
rectus.sql"# dump DB to file
ls -lh"$BASE/backups/directus.sql"# confirm backup exists
```

If you want, paste **your OS** (Ubuntu version / WSL yes/no). I'll adjust the install block (Docker repo vs snap vs Debian) to match exactly so you don't hit package errors.